UNIVERSITY OF OSLO
Department of Informatics

Class Modeling of OSLC Resources

Research report 428

Weiqing Zhang

ISBN 82-7368-392-3
ISSN 0806-3036

May 29, 2013

Class Modeling of OSLC Resources

Weiqing Zhang

Department of Informatics, University of Oslo, Norway
weiqingz@ifi.uio.no

Abstract. This report investigates the possibilities of modeling OSLC Resources with Class Models, and also transforming these models into OSLC server and client code fragments that use these resources.

Keywords: Class Model; OSLC;

1 Introduction

1.1 Background

In software development tool integration is done based upon the important notion of adapters that are standardized alternatives to tool-specific APIs. Much effort has been put into specifying adaptors based upon integration scenarios, but less effort into identifying potential common concepts, which collect certain common software lifecycle properties and represent integrated tool models in a uniform way. These common concepts may be in integration scenarios like specifying trace relationships between elements of different model. Furthermore, an investigation is needed on how to deal with non-common, tool-specific concepts which can not be covered by tool-independent concepts. For instance, a higher-level service might make use of certain information in case it is provided by a specific tool. Or this service might only need a subset of information of the common concepts.

Tools can be loosely coupled via a set of services, or web services applied on standardized communication protocols (Web Service Oriented Architecture). A recent framework provided by the *OSLC* community [15] finds an agreement among the stakeholders on specification for tool integration. The key concepts are a uniform access to shared resources, a common vocabulary/formats, and a loose coupling approach between tools through REST architectures [5]. A tool integration project in which we participated, named iFEST [9], has adopted OSLC specifications. It implies that adaptors work on representatives of the real artifacts of tools in terms of artifact resources, and that these have to be specified in terms of OSLC-tables of properties (OSLC specifications). In addition, adaptors also specify services. A tool is integrated by implementing such adaptor specifications. The project also aims to support the generation of the part of adaptor implementations (server and client code) that adhere to the OSLC specification.

1.2 Problem Statement

Many of the tools that are to be integrated are modeling tools that are based upon (explicit or implicit) metamodels that define the languages in which models are made. Engineers that are supposed to integrate a tool will therefore have the metamodels of the languages supported by the tools. A typical case is the commonalities between a set of tools are captured in a common language and therefore by a common metamodel.

OSLC specifications as such defines a small set of common properties that are common among all resources. In addition there are often some project-specific properties. It is believed that tool integration will benefit from being based upon common models and data in between the minimal properties-approach and a full (unrealistic) common model between all tools. Integration will then be based upon different tools in different parts of the lifecycle both contributing to these common models (or fragments of models), and using common concepts in their specific models.

Using models brings benefits like raising abstraction level, promoting communication, capturing domain knowledge, etc.. With MDE technology, it also supports generation of design and implementation artifacts, such as corresponding OSLC specifications and fragment codes in above case. Then an interesting question is raised, that is how to model the OSLC resources with appropriate approaches, and generate implementation artifacts.

1.3 Objectives

The objectives of this investigation is therefore.

- to investigate if both the OSLC specification and the implementation can be generated from a model (or a subset of a model) of what is the basis for integration, and
- to investigate which other OSLC-based open-source web frameworks can be leveraged in order to generate an OSLC specification and language-specific client/server skeletons running out of the box on a certain open-source infrastructure. This approach should support generation for different languages in a straight-forward way.
- to identify candidates for common (fragments of) models and common concepts that are used by different tools.
- to identify search concepts in order to traverse the received data. This approach is needed in case of a coarse-grained client/server data interface exchanging a large part of the model instance data.
- to identify common methodologies and transformation concepts to deal with the transformation of any subset of a models to OSLC, to client/server skeletons of a certain language (e.g. Java, C Sharp, Python) and any glue code needed to get it running on an OSLC-based web framework.

The investigation should start out with a model of what to integrate (based upon a model or metamodel), and then generate both the OSLC specification

and code fragments. Given an OSLC specification it is also possible to manually make a corresponding model from which the implementation may be generated. The investigation should give feasible modeling guideline based upon examples, also with tools that support potential integration approaches.

2 The Class Modeling Approach

2.1 Introduction

As a standardized general-purpose modeling language in the field of object-oriented software engineering, it is common to use UML class model to define integrated data. It should be made clear up front that the investigations are not made for using class models for (technology-independent) representatives of real artifacts for the purpose of tool integration (and tool adaptors working on such representatives). Instead, the investigation is on the use of class models as models of OSLC resources as representatives of real artifacts. The reason for making this clear is that class models and OSLC models are so different in nature that a class model of OSLC resources can never be a perfect match.

Still, there is a good reason for doing a second-best try: class models are well-known by those who are going to make resource specification, experience has shown that making class model of OSLC resource shapes may reveal errors in the OSLC specifications, and resource shape specifications are usually based upon a metamodel (which is a special kind of model) that defines the language in which artifact model elements are made - and the resources are to represent these model elements.

The investigation could be based upon any representative of a class modeling tool, e.g. UML tool, as the reason that models are used rather than metamodels. Still, Eclipse Modeling Framework is used with Ecore, a subset of MOF, which in turn is a subset of UML. The reason for this is that this is rather widespread and widely used for making metamodels on which our resource models are based.

In OSLC, resources are representatives of models and model elements. A *resource shape* defines the set of OSLC Properties expected in a resource for specific operations (i.e. creation, update or query) for each their value types, allowed values, cardinality and optionally. Models are made in languages defined by metamodels - this includes models for which there may be no concrete syntax, as e.g. requirements models. Resources are elements according to models - not according to metamodels:

- OSLC resources are instances according to resource shapes defined by resource definitions,
- If defined by a class model, resources would be objects of classes

There are two things to model by means of class models:

- Properties, and in OSLC these are defined independently of classes that model resource shapes
- Resources shapes

2.2 Overview of Approaches

Three different approaches are investigated, with their advantages and disadvantages. All the approaches model resource shapes by means of classes, and properties by means of attributes of these classes, even though in principle classes of resources in OSLC are not the same classes of objects in a class model.

In order to have a better understanding of abstractions in different MOF levels, models (here UML-models), OSLC specifications and Java are compared. As shown in figure 1, resources corresponds to objects of classes. This will also be reflected in the Java code that will be generated: it will have classes corresponding to resource shapes; resources will be objects of these classes, and they will have operations for being communicated in the form of e.g. XML.

Level	Model-based	OSLC	Java
Meta-meta	MOF		EBNF
Language (metamodel)	UML	RDF	Java grammar
Models	UML models, with classes	Resource (shape) definitions	Java programs
Execution	Model executions (in case of executable UML, with objects of classes), or just objects of classes	Exchange (between servers and clients) of resource (instance)s according to resource (shape) definitions	Program executions with objects

Fig. 1. Resources Corresponds to Objects of Classes

An OSLC resource is managed by an OSLC Service. OSLC uses a simple model of resources with property values in the Resource Description Framework (RDF) data model. OSLC also builds upon the XML namespace mechanism.

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
dcterms:conformsTo	Exactly One	True	String Literal	N/A	N/A	Indicates the established standard to which the described resource conforms.
dcterms:format	Zero or One	True	String Literal	N/A	N/A	The format of the data, e.g. XML, zipped XML, base64 encoded zipped XML, etc. Recommended best practice is to use a controlled vocabulary such as the list of Internet Media Types.
dcterms:source	Zero or One	True	Resource or String Literal	Reference or Inline	N/A	The byte content of the data. If provided inline this value will be a string literal, otherwise it will be a URI reference to a resource.
ifcore:encoding	Zero or One	True	String Literal	N/A	N/A	Indicates the character encoding of the data when it is textual, e.g. UTF-8, ASCII, etc.

Fig. 2. File Descriptor Resource Shape

Project-specific common concepts like resource shapes have been defined [10], such as "FileDescriptor" (figure 2), "ResourceDescriptor" (figure 3), and "Ele-

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
ifcore:elementType	exactly-one		String	n/a		Type of the element must of the possible element types of the resource. For example, a ChangeManagement resource can be actually a change request, or a BugRequest. These are 2 different types.
dcterms:label	zero-or-one		String			A textual description to display to its user.
ifcore:resource	exactly-one		Resource	Reference	Resource	The reference to the actual resource being wrapped.

Fig. 3. Resource Descriptor Resource Shape

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
ifcore:container	exactly-one		Resource	Reference	FileVersion FileDescriptor QueryCapability	The reference to the container of the element, could be either a FileVersion, iFESTFile or a QueryCapability.
ifcore: elementIdentifier	exactly-one		String	n/a		The identifier of the element refered to could be an URI Fragment, ObjectID or other unique identifier of the element inside the container.
ifcore: elementType	exactly-one		String	n/a		Type of the element. Must be one of the possible elements contained in the container.
dcterms:label	zero-or-one		String			A textual description to display to its user.
Child1	zero-or-many		Resource	Inline	ElementDescriptor	The potential set of children element Descriptors that this resource contains.

Fig. 4. Element Descriptor Resource Shape

mentDescriptor" (figure 4). These are examples on what kind of OSLC specifications that it shall be possible to represent by (and generate from) a resource class model.

An OSLC resource shape is specified by means of a table, each row specifying a property of the corresponding resources.

These are the three approaches:

- Approach 1: Using Ecore features as far as possible
- Approach 2: Independent modeling of Properties by EClasses
- Approach 3: Independent modeling of Properties by EDataTypes

2.3 The Approaches

Approach 1: Using Ecore features as far as possible The OSLC resource properties can be represented directly by the properties of attributes in of Ecore class models. A general overview of the mapping rules between OSLC concepts and Ecore models is given in figure 5.

Figure 6 illustrates how resource shapes are modeled with Ecore class models, while figure 7 shows the details of how to specify each data value property with EAttribute properties.

Traceability adaptor specification specifies how the source and target properties of TraceRelationship can trace to any resource of FileDescriptor, ElementDescriptor, and ResourceDescriptor. This is modeled in figure 8.

OSLC Resource Shape Level		
OSLC Recourse Shape		**Ecore Model**
Resource Shape		Ecore\EClass
Data value property		Ecore\EAttribute
Reference property		Ecore\ EClass\EReference
OSLC Resource Property Level: Data value property by EArritbute		
value for OSLC properties		**Ecore properties**
Prefixed Name		Ecore \ EAttribute\ Extended Metadata\Namespace
Occurs		Ecore\ EAttribute\ Advanced \ Lower Bound\ Upper Bound
Read-only		Ecore\ EAttribute\ Advanced \ Changeable\ Unsettable
Value-type		Ecore\ EAttribute\ Advanced \EType\ EAttribute
Representation		Ecore\ EAttribute\ Extended Metadata \Representation
Range		Ecore\ EAttribute\ Extended Metadata \Range
Description		Ecore\ EAttribute\GenModelDoc
OSLC Resource Property Level: Reference Property by EReference		
value for OSLC properties		**Ecore properties**
Prefixed Name		Ecore\ EClass\EReference\Extended Metadata\Namespace
Occurs		Ecore\ EClass\EReference\ Lower Bound\ Upper Bound
Read-only		Ecore\ EClass\EReference\Advanced\ Changeable\ Unsettable
Value-type		Ecore\ EClass\EReference\Advanced \EType
Representation		Ecore\ EClass\EReference\Extended Metadata \Representation
Range		Ecore\ EClass\EReference\Extended Metadata \Range
Description		Ecore\ EClass\EReference\GenModelDoc

Fig. 5. Mapping Rules in Approach 1

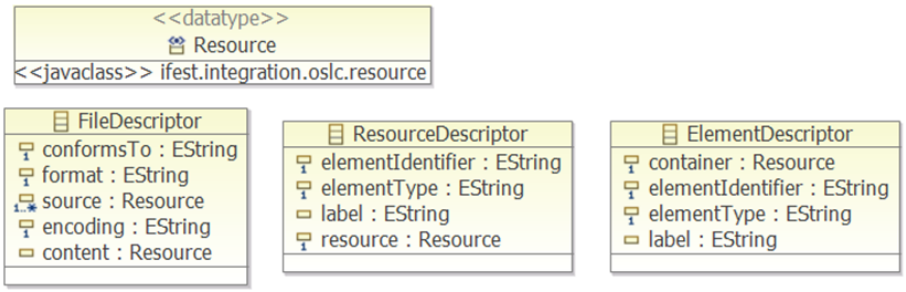

Fig. 6. Resource Shapes in Approach 1

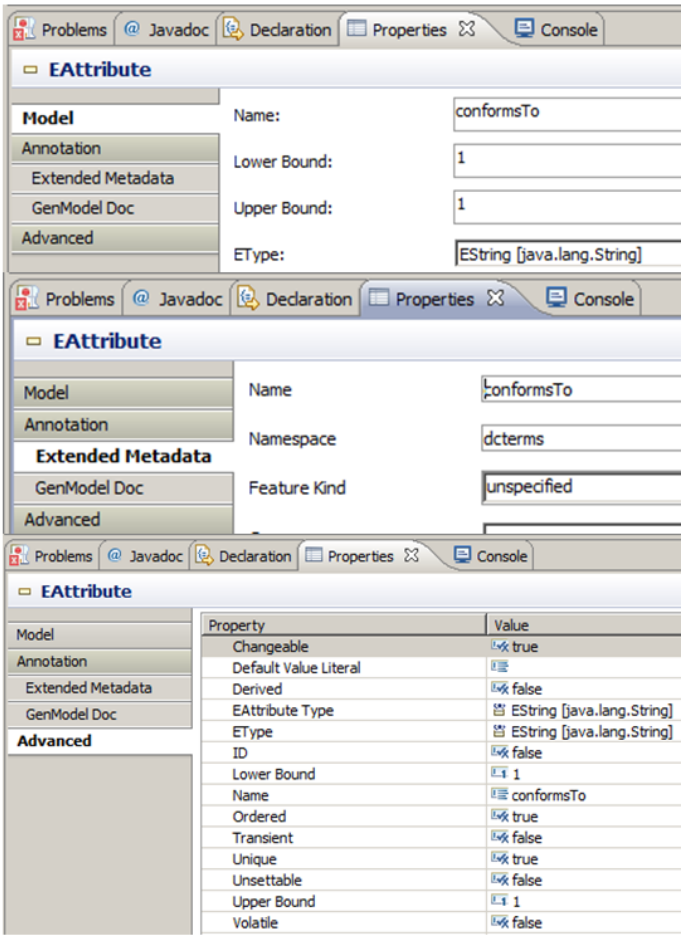

Fig. 7. Specify Data Value Properties in Approach 1

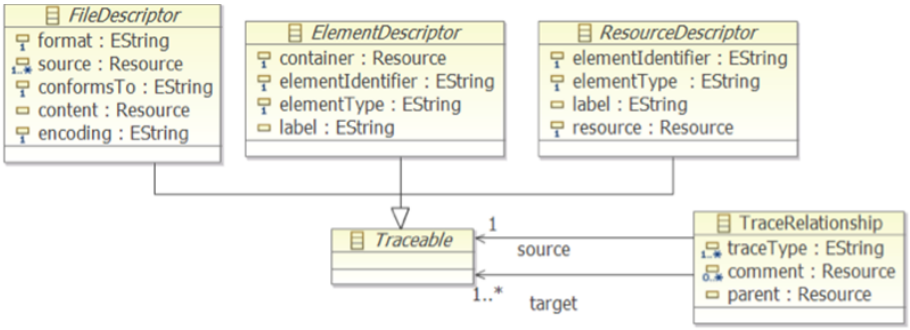

Fig. 8. Traceability Model in Approach 1

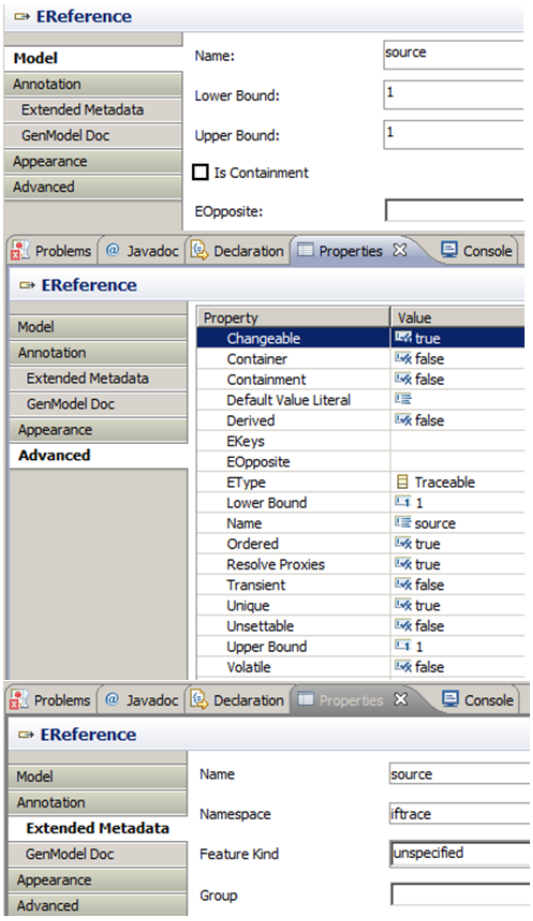

Fig. 9. Specify Traceability Model Property Values in Approach 1

More details are given in figure 9: EReference is used to represent the 'source' and 'target' properties of TraceRelationship; while the values of these properties are represented by EReference.

Figure 10 is a complete class model of the resources defined in adaptor specifications, including requirement, traceability and iFEST common properties.

The advantage of this approach is that it simply uses the EAttribute properties provided by EMF Tools. This is the most intuitive and directly way to make class models with EMF.

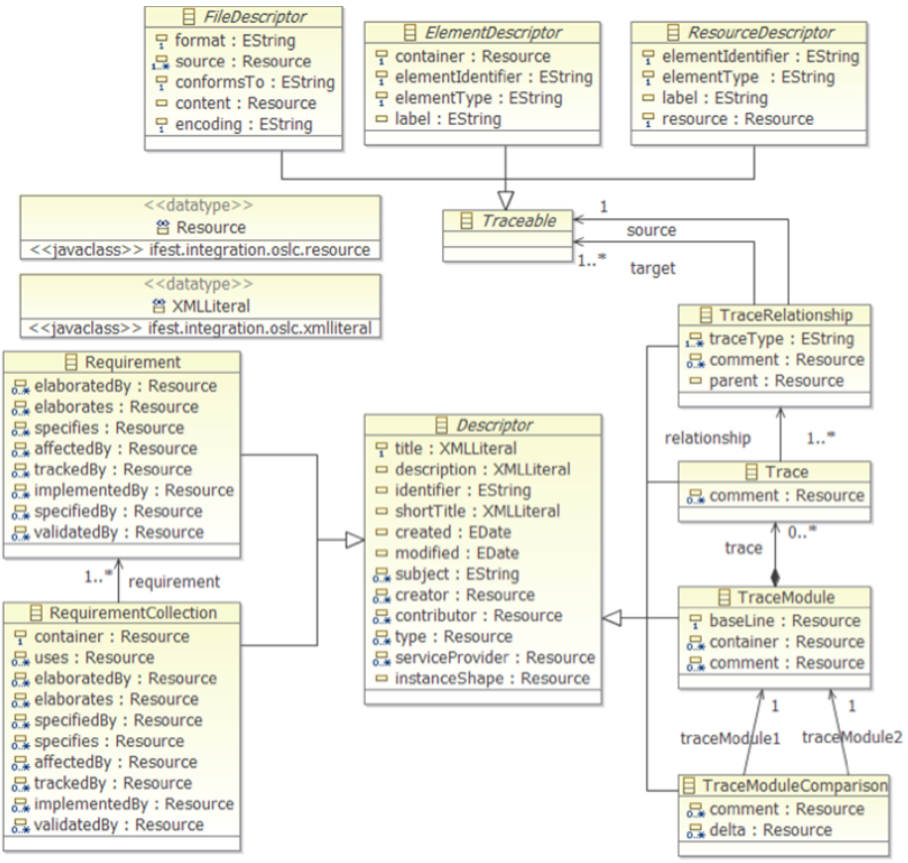

Fig. 10. Completed Class Model for OSLC Resources in Approach 1

There are also several disadvantage of this approach:

– Duplicate definitions: The properties (such as 'conformsTo' and 'format' from FileDescriptor) are defined together with their corresponding resource

EClass. If these properties are also properties of other resource EClasses, the properties need to be specified again for these other resources.
– Vulnerable to changes: As the same properties are defined within different resource, it needs to apply the change in many places when these properties need to be updated.

In order to overcome the above disadvantages, the following two approaches separate the definition of resources and the definition of Properties, to make the approach more flexible. This is done by modelling the OSLC Properties either by EClass or by EDataType.

Approach 2: Independent modeling of Properties by EClasses In approach 2, the definitions of resource shapes and properties are separated.

Properties for different specifications are defined separately. The specifications include namespaces like ifcore, iftrace, oslc, oslcRm, dcterms, etc. The properties of OSLC resources are modeled by EClasses.

The detailed mapping rules are given in figure 11.

OSLC Resource Shape Level		
OSLC Shape Recourse		**Ecore Model**
Resource shape		Ecore\EClass
Property of Resource defined by Resource Shape		Ecore\EClass
OSLC Resource Property Level		
value for OSLC properties		**Ecore properties**
Prefixed Name		Ecore\ by default it is the name of ecore file
Occurs		Ecore\ EClass \ EAttribute, Value of "Occurs" is specified with Default value Literal of EAttribute
Read-only		Ecore\ EClass \ EAttribute, Value of "Read-only" is specified with Default value Literal of EAttribute
Value-type		Ecore\ EClass \ EAttribute,
Representation		Value of "Representation" is specified with Default value Literal of EAttribute
Range		Ecore\ EClass \ EAttribute, Value of "Range" is specified with Default value Literal of EAttribute
Description		Ecore\ EAttribute\GenModelDoc

Fig. 11. Mapping Rules in Approach 2

As show in figure 12, the ifcore namespace is a package with the properties modeled by EClasses.

The value of a property is represented by the "Default Value Literal" of the corresponding EAttribute. Figure 13 is an example shows how to specify the value of "readOnly" from ElementIdentifier.

The resource class model then uses these EClasses for their properties, with the details illustrated in figure 14, figure 15, figure 16.

The main advantage of this approach is that it captures the fact that OSLC properties are defined independently of resource shapes. It may be regarded as a disadvantage that the notation for resource shapes having properties is by means

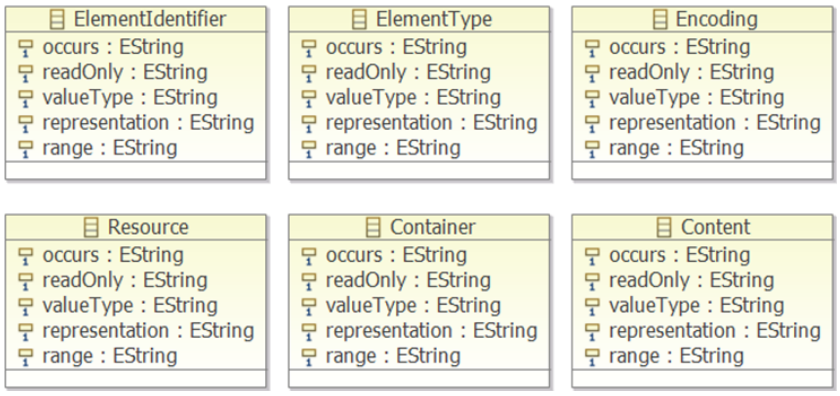

Fig. 12. Model ifcore Properties in Approach 2

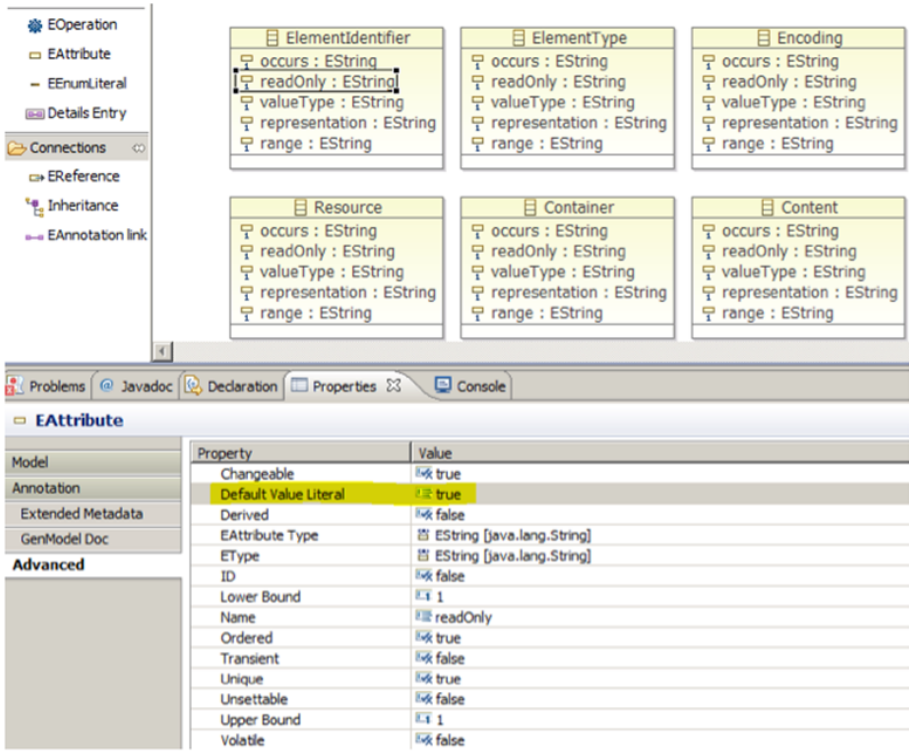

Fig. 13. Specify Data Value Properties in Approach 2

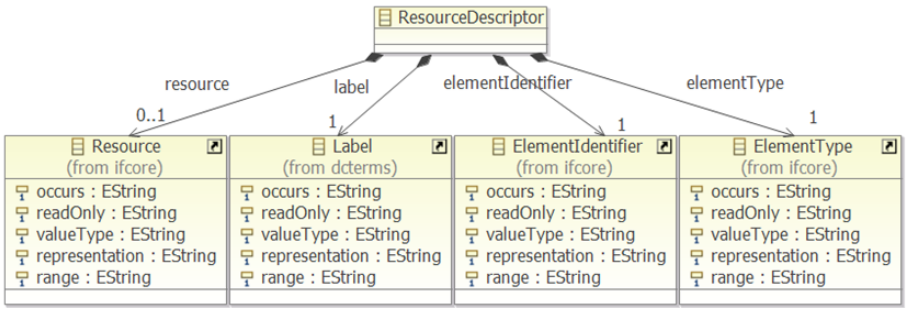

Fig. 14. ResourceDescriptor Class Model in Approach 2

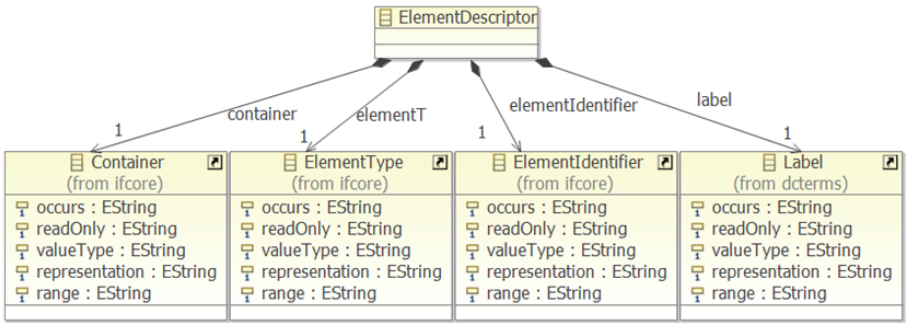

Fig. 15. ElementDescriptor Class Model in Approach 2

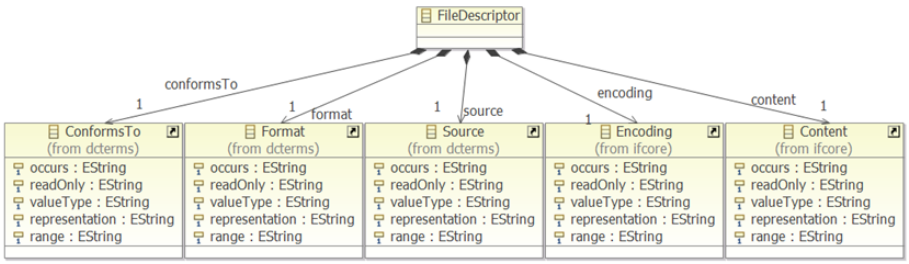

Fig. 16. FileDescriptor Class Model in Approach 2

of associations (see e.g. figure 16). However, it reflects the notation that is used for RDF models, except that they do not use composition but pure associations.

Approach 3: Independent modeling of Properties by EDataTypes In this approach properties are modeled by means EDataTypes. The detailed mapping rules are given in figure 17.

OSLC Resource Shape Level		
OSLC Recourse Shape		**Ecore Model**
Resource shapes		Ecore\EClass
Property of Resource that defined by Resource shape		Ecore\EDataType
OSLC Resource Property Level		
value for OSLC properties		**Ecore properties**
Prefixed Name		Ecore\ by default it is the name of ecore file
Occurs		Ecore\ EDataType \ Annotation\ new Annotation "property"\ key "Occurs" with corresponding value
Read-only		Ecore\ EDataType \ Annotation\ new Annotation "property"\ key "Read-only" with corresponding value
Value-type		Ecore\ EDataType \ Annotation\ new Annotation "property"\ key "Value-type" with corresponding value
Representation		Ecore\ EDataType \ Annotation\ new Annotation "property"\ key "Representation" with corresponding value
Range		Ecore\ EDataType \ Annotation\ new Annotation "property"\ key "Range" with corresponding value
Description		Ecore\ EDataType \ Annotation\GenModelDoc

Fig. 17. Mapping Rules in Approach 3

Figure 18 shows how ifcore properties are modeled with EDataTypes.

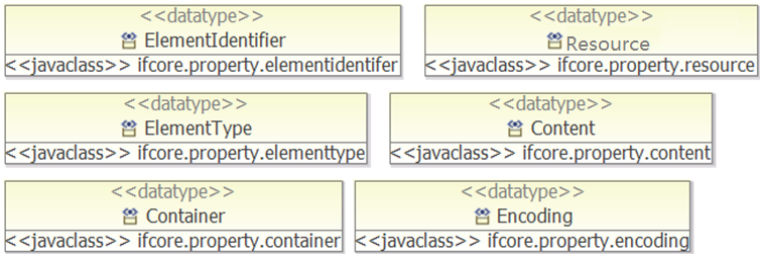

Fig. 18. Model ifcore Properties in Approach 3

Figure 19 shows how values are stored in the ElementIdentifier property.

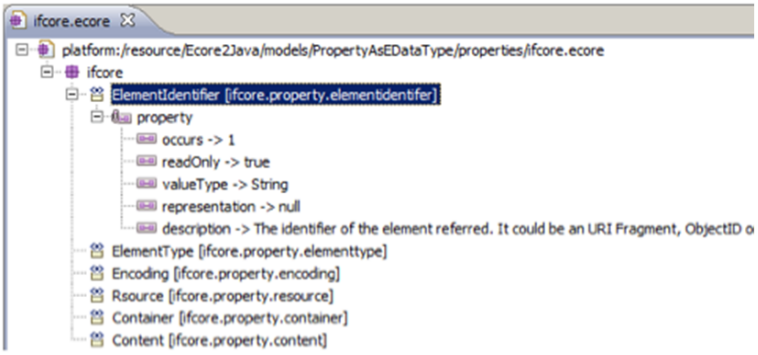

Fig. 19. Specify Data Value Properties in Approach 3

In the resource Ecore model, the above-defined EDataTypes are assigned as the ETypes of the corresponding resource property. The assignment detail is shown in figure 20.

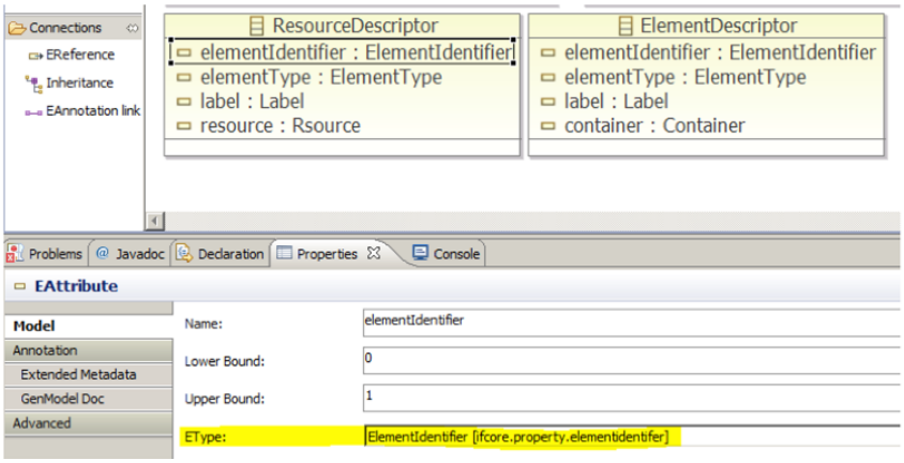

Fig. 20. Assign Values to Resource Properties in Approach 3

Figure 21 shows how an Ecore class model for OSLC resource shapes in approach 3.

The only advantage of this approach is that the notation for properties is as you would expect it to be for class models. The only reason for considering this approach is, however, due to a problem with EFM. In proper UML it is possible to define an attribute with a type that is a user-defined class, while

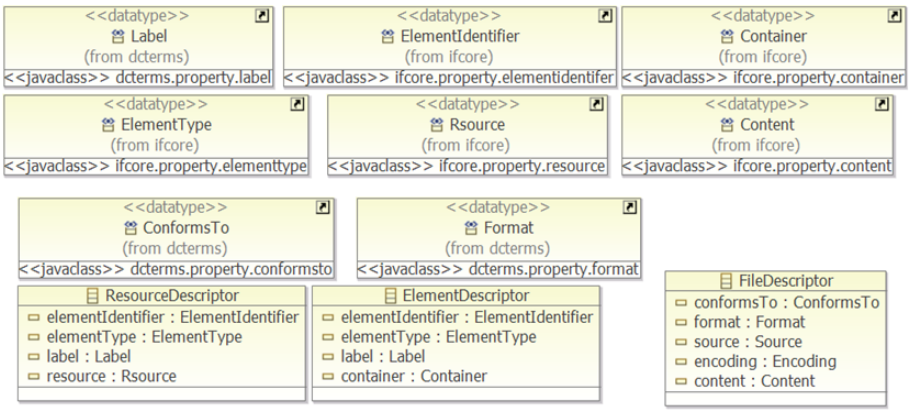

Fig. 21. Ecore Class Model for OSLC Resource Shapes in Approach 3

types of attributes in EFM have to be datatypes, somewhat more limited than classes. With proper UML it would therefore be possible to combine the last two approaches: define the property (types) by classes and define the properties of resources type as attributes with these classes as types - and still have the notation of the last approach.

3 Code Generation

The class models can be transformed into OSLC resources and corresponding OSLC client/ server code.

3.1 Model Transformation

The integration models are transformed into Web services through M2T (model to text) transformations.

The M2T transformation languages express transformations which transform a model into text, for example a platform-specific model into source code or documentation. This can be achieved by MOFScript [14] or MOFM2T (MOF Model to Text Transformation Language) [13]. MOFScript is a M2T transformation tool that supports generation of implementation code or documentation from models. It provides a metamodel-independent language that allows to use any kind of metamodel and its instances for text generation. The MOFScript tool is based on EMF and Ecore as metamodel framework. The alternative MOFM2T is part of OMG's Model-driven architecture (MDA) and reuses many concepts of MOF such as OMG's metamodeling architecture. For this reason the implementation of MOFM2T fully depends on OMG resources.

The MDA provisioning engine ModelPro [12] is open source and able to read models and "provision" source code, documents web pages automatically. ModelPro is integrated with MagicDraw UML with the Cameo SOA+ Plug-in, which

automates the design and development of Service Oriented Architectures, initially on Java Platforms. With SoaML and ModelPro, service oriented architectures can be up and running quickly and efficiently, ready to integrate new and existing capabilities into coherent solutions based on the requirements, services and processes.

The OSLC servers provide tool resources through web services. The Artifact and Role classes (in EMF models) are the basis for generating OSLC resource and tool adaptor server source code through transformation.

Web services are established based on the generated JEE Annotations. The JEE Annotation is a form of syntactic metadata that can be added to Java source code of JEE frameworks. It can be embedded in class files generated by the compiler and may be retained by the Java VM to be made retrievable at run-time.

The generated OSLC web services code is based upon the OSLC Eclipse LYO [16] project. It adopts OSLC specifications and builds OSLC-compliant tool adaptor server and client. The code is based upon RDF, Web Service technology. As depicted in figure 22, Java annotation and JAX-RS method code are generated from EMF model, which simplify the development process.

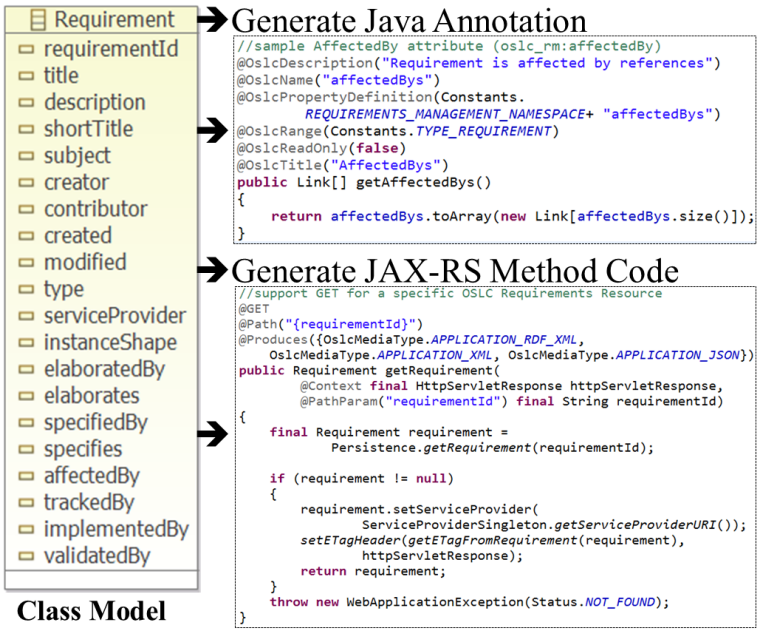

Fig. 22. Code Generation

3.2 Invoke Functionalities from Web service

In the service-base server/client implementation, the identified functions and service points are invoked from servers. Each server provides its functionalities together with OSLC resource through web service.

There are three main components of Web services in this architecture: *service provider*, *service requester*, and *service broker*. When service client requests a service, the service broker will check its WSDL service registry. Once the service is found, the service broker will generate SOAP request message and send to the service provider. The service to be deployed on the Web server exposes a number of methods callable from the SOAP client. The service residing on the Web server acts as a client to a integrated tool that resides on a different server. The development tool has a web service wapper on the web server. This will facilitate a clean separation from the original tool calls. A Web service adapter (wrapper) is built over the integrated tools to expose it as Web service. The development tool customized APIs can be isolated with this kind of design. The wrapper invokes the methods from the integrated tools through SOAP.

The services-based tool chain invokes the functionalities from the integrated tools like trace capability or transformation capability in this step. The tool clients then can invoke the functionalities from other integrated tool adaptor servers.

The tool adaptor server includes the common definitions used for the adaptor, such as resource definitions (compliant to Artifact class and Role class) and constants. It mainly includes three components. The *OSLC Registry* web application is used as an OSLC Catalog for the service providers. When each application starts, it registers its OSLC service provider details with OSLC Registry. The *OSLC Wink* is the framework that builds RESTful Web services. The *OSLC Provider* provides OSLC resources in RDF or Json.

A typical tool adaptor service, such as a service to retrieve a number of UML model elements from a UML Rhapsody, would look like:

umlArtifact[*] getUML(String ArtifactUID)

The umlArtifact instances are used as return parameters. The real UML model elements are contained in the returned OSLC RDF file. The above adaptor integration service is mapped to corresponding OSLC service, such as:

HTTP GET *http://umlServer:8080/umlArtifact/UID/all*

With the Artifact UID property, it can easily locate the corresponding Artifact (OSLC resource), and then access real model element. The adaptor server provides services and receives requests from consumers to manipulate the OSLC resource through HTTP methods (GET, PUT, DELETE, and POST). There is client code acts as service consumer to test the services provided by adaptor server.

4 Related Work

OSLC resources are defined and represented in various kinds of forms, such as RDF/XML, XML, Turtle, JSON and Atom. We haven't found other researches

work on direct transformation between OSLC concepts and UML class model concepts. However, several works on transformation between these represented forms and UML models, relational data models or EMF have been tried.

[3] proposed an interoperable data model called R2iDM, which is closely designed to match the graphic RDF triple structure. The models have the capability to general an RDF triple statement automatically with minimal intervention of an data modeler. Relational Metadata concept [11] is designed based upon XML and Entity Relationship model, with the purpose of retrieving relational information in WEB contents. The goal of Relational Metadata is to provide the WEB contents creators with easy control over metadata and with automatic RDB integration. Relational Metadata is compatible with RDF therefore conversions between them are feasible. In [7], the object domain model is mapped to an ontology and to a persistence model, with the mapping rules specified by a model annotation performed by a domain expert. The approach automates the processes and generates data transformations via model weaving techniques.

[4] described the mapping between RDF and any ontology modelling language that has its abstract syntax defined using the OMG's meta-object facility (MOF) model. The solution defined transformations between other types of model (such as UML) and ontologies, between different ontology modelling languages, or to modify ontologies without changing the language.

[8] presents an transformation solution between EMF objects, and RDF resources through the ATL model transformation language. It provides a prototype that offers a small Java library for the instantiation and serialization of EMF objects from, and to RDF resources.

[1] presents an approach that builds a bridge between a UML model and ontology OWL (Web Ontology Language) through Eclipse platform. Similar to our approaches, the transformation is based on transformation rules which make it possible to connect the concepts of UML and OWL. [6] shows an architecture that consists of Ontology Definition Metamodel defined using Meta Object Facility (MOF), based on the OWL, as well as the related Ontology UML Profile (OUP). It also shows the transformation from OUP definition such as XMI into OWL description.

[2] presents a similar code generation research work on transformation from formalized OSLC specifications to specialized code. The specifications systematize tool integration and the generated code automate the process of building tool adapters.

5 Conclusion

Three different approaches for modeling OSLC Resource are proposed and compared in this report.

The first approach, with the direct use of EMF mechanisms, is the one that is easiest to use for established users of EMF. The main reason for investigating the two last approaches is that that properties and resources defined by OSLC

are subject to change, either because the specifications are not mature or because of changes due to standardization efforts. In OSLC, properties are defined independently of the specification of resources with these properties. In the last two approaches it will be possible to make a class model of such properties, also independent of any class models of resources, and they are therefore more true to the OSLC-way of specifying things. The design and implementation artifacts like Web Service code fragments then can be generated from these models.

Acknowledgment

The author would like to thank for the support given by ARTEMIS Tool Integration project iFEST which enabled us to gain expertise in the domain of embedded system development and tool integration. Birger Møller-Pedersen, Matthias Biehl and Philippe Soulard give great helps on completing this report.

References

1. A. Belghiat and M. Bourahla, *Transformation of uml models towards owl ontologies*, Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), 2012 6th International Conference on, 2012, pp. 840–846.
2. M. Biehl, J. El-Khoury, and M. Torngren, *High-level specification and code generation for service-oriented tool adapters*, Computational Science and Its Applications (ICCSA), 2012 12th International Conference on, 2012, pp. 35–42.
3. Mi-Young Choi, Chang-Joo Moon, Doo-Kwon Baik, Young-Jun Wie, and Joong-Hee Park, *Interoperability between a relational data model and an rdf data model*, Networked Computing and Advanced Information Management (NCM), 2010 Sixth International Conference on, 2010, pp. 335–340.
4. Stephen Cranefield and Jin Pan, *Bridging the gap between the model-driven architecture and ontology engineering*, Tech. report, 2005.
5. Roy Thomas Fielding, *Architectural styles and the design of network-based software architectures*, Phd thesis, University of California, 2000.
6. D. Gasevic, D. Djuric, V. Devedzic, and V. Damjanovic, *From uml to ready-to-use owl ontologies*, Intelligent Systems, 2004. Proceedings. 2004 2nd International IEEE Conference, vol. 2, 2004, pp. 485–490 Vol.2.
7. Guillaume Hillairet, Frdric Bertrand, and Jean Yves Lafaye, *Mde for publishing data on the semantic web.*
8. Guillaume Hillairet, Frédéric. Bertrand, and Jean-Yves Lafaye, *Bridging EMF applications and RDF Data Sources*, 4th international workshop on Semantic Web Enabled Software Engineering (SWESE) at ISWC'08 (Karlsruhe, Germany), October 2008, p. 26 (English).
9. iFEST Community, *iFEST - industrial Framework for Embedded Systems Tools*, ARTEMIS-2009-1-100203, 2010.
10. iFEST Project, *iFEST Common Properties and Resources* , iFEST, 2013.
11. A. Imai and S. Yukita, *Rdf model and relational metadata*, Advanced Information Networking and Applications, 2003. AINA 2003. 17th International Conference on, 2003, pp. 534–537.
12. ModelDriven.org, *SoaML/ModelPro Tutorials. SoaML Cartridge.*, http://portal.modeldriven.org/project/soamlcartridge, 2009.

13. Object Management Group, *MOF Model To Text Transformation Language, 1.0*, http://www.omg.org/spec/MOFM2T/1.0/.
14. Object Management Group (OMG), *MOF Model to Text Transformation*, OMG Document ad/05-05-04.pdf .
15. OSLC Project, *OSLC - Open Services for Lifecycle Collaboration Core Specification Version 2.0* , http://open-services.net/bin/view/Main/OslcCoreSpecification, 2011.
16. The Eclipse Foundation, *Eclipse Lyo Project*, http://www.eclipse.org/lyo/, 2012.